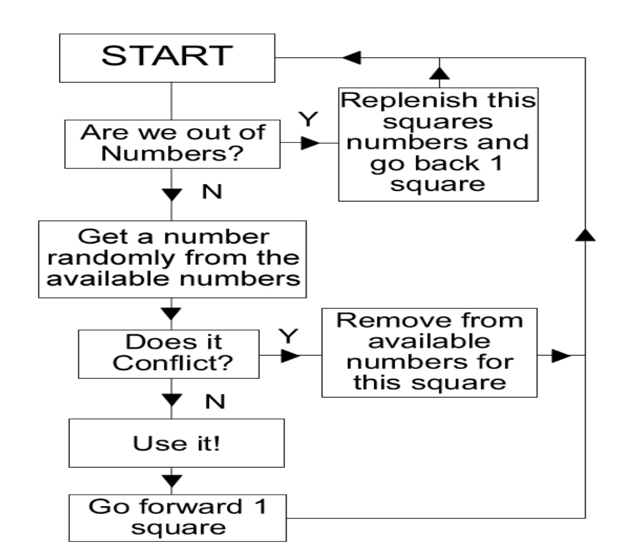
بسم الله الرحمن الرحيم

خوارزميات الحاسوب



أعداد: احمد فتح العليم عبيد الله

ahmed_first_222@hotmail.com

1-1 مقدمة:

الخوارزمية هي مجموعة من الخطوات الرياضية والمنطقية المتسلسلة اللازمة لحل مشكلة ما. وسميت الخوارزمية بهذا الاسم نسبة إلى العالم المسلم الطاشقندي الاصل أبو جعفر محمد بن موسى الخوارزمي الذي ابتكرها في القرن التاسع الميلادي. الكلمة المنتشرة في اللغات اللاتينية والأوروبية هي «algorithm» وفي الأصل كان معناها يقتصر على خوارزمية لثلاثة تراكيب فقط وهي: التسلسل (Sequence) والاختيار (selection) والتكرار (looping).

- التسلسل: تكون الخوارزمية عبارة عن مجموعة من التعليمات المتسلسلة، هذه التعليمات قد تكون إما بسيطة أو من النوعين التاليين.

- الاختيار: بعض المشاكل لا يمكن حلها بتسلسل بسيط للتعليمات، وقد تحتاج إلى اختبار بعض الشروط وتنظر إلى نتيجة الاختبار، إذا كانت النتيجة صحيحة تتبع مسار يحوي تعليمات متسلسلة، وإذا كانت خاطئة تتبع مسار آخر مختلف من التعليمات. هذه الطريقة هي ما تسمى اتخاذ القرار أو الاختيار.

- التكرار: عند حل بعض المشاكل لا بد من إعادة نفس تسلسل الخطوات عدد من المرات. وهذا ما يطلق عليه التكرار.

و قد أُثبت أنه لاحاجة إلى تراكيب إضافية. استخدام هذه التراكيب الثلاث يسهل فهم الخوارزمية واكتشاف الأخطاء الواردة فيها وتغييرها.

2-2 **تعريف الخوارزمية:**

الخوارزمية , تمت تسميتها بهذا الاسم بعد القرن التاسع بواسطة العلامة جعفر بن موسى الخوارزمي ..

وتعرف الخوارزمية بالاتي:

- الخوارزمية هي عبارة عن مجموعة من القوانين لتنفيذ عملية حسابية أما عن طريق اليد أو الآلة.
- الخوارزمية هي عبارة عن مجموعة من الخطوات المنتهية لتحقيق النتيجة المطلوبة.
- الخوارزمية هي عبارة عن سلسلة من الخطوات الحسابية لتحويل المدخلات إلى مخرجات.
- الخوارزمية هي سلسلة من العمليات التي تجري على البيانات التي يجب أن تكون منظمة في صورة هياكل بيانات.
- الخوارزمية هي عبارة عن فكرة مجردة لبرنامج ليتم تنفيذه في آلة فيزيائية (الحاسوب نموذجا).

أشهر خوارزمية في التاريخ في زمن الإغريق هي خوارزمية إقليدس لحساب القاسم المشترك الأعظم لعددين, ظهرت هذه الخوارزمية كحل للأطروحة II في الكتاب VII لإقليدس "Elements " الذي يتألف من ثلاثة عشر كتابا . ويحتوي على عدد 465 أطروحة .

2 3 **خوارزمية الضرب التقليدية :**

1-3-3 الضرب , بالطريقة الأمريكية:

اضرب العدد في المضروب وحد تلو الآخر لكل رقم من اليمين إلى اليسار

```
      9 8 1
    1 2 3 4
-----------------
    3 9 2 4
  2 9 4 3
1 9 6 2
9 8 1
-----------------
1 2 1 0 5 5 4
```

2 3 2 الضرب , بالطريقة الإنجليزية :

اضرب العدد في المضروب وحد تلو الآخر لكل رقم من اليسار إلى اليمين

```
        9 8 1
      1 2 3 4
-----------------
    9 8 1
    1 9 6 2
      2 9 4 3
        3 9 2 4
-----------------
  1 2 1 0 5 5 4
```

- الخوارزميات هي عبارة عن فرع من فروع علوم الحاسوب يتعلق تصميم و بتحليل الخوارزميات.
  - في التصميم يتعلق بـ :
    - i. وصف الخوارزمية على المستوى المجرد باستخدام اللغة الزائفة pseudo code.
    - ii. إثبات صحة الخوارزمية ,بحيث تعطي الخوارزمية حل للمشكلة في جميع الحالات.
  - التحليل يتعلق بتقييم الأداء (تحليل التعقيد) .

4-2 خطوات حل أي مشكلة بواسطة الحاسوب:

- تعريف المسالة وتحليلها(problem definitions &analysis).
- وضع خوارزمية الحل (algorithm).
- كتابة البرنامج بإحدى لغات البرمجة(writing the program).
- ترجمة البرنامج إلى لغة الآلة(compilation).
- تنفيذ البرنامج (execution).

2-4-1 تعريف المسالة وتحليلها

يتم تحديد أبعاد المسالة, والأهداف المطلوب الوصول إليها عن طريق الاتى:-

- تعريف المخرجات وشكلها بدقة(النتائج المراد تحقيقها من المسالة).
- تحديد المدخلات بناء المخرجات المطلوبة.
- تحديد طرق الحل المختلفة وتقيمها لاختيار أفضلها من حيث السهولة و سرعة التنفيذ والمساحة التي تحتاجها من الذاكرة.

2-4-2 وضع خوارزمية الحل

بعد اختيار الطريقة الأحسن لحل يتم التعبير عن هذه الطريقة بالخوارزمية.

2-4-3 كتابة البرنامج

في هذه الرحلة يتم التعبير عن الخوارزمية بإحدى لغات البرمجة(الكود).

2-4-4 مرحلة الترجمة

وفية يتم تحويل البرنامج إلى لغة الآلة حسب نوع لغة يتم استخدام المترجم أو المفسر,وتمر عملية الترجمة بمراحل وهى :-

- مرحلة التحليل اللغوي والصرفي :
  ويتم فيه مطابقة التوكن برنامج الهدف مع القواعد اللغوية للغة المستخدمة واكتشاف الأخطاء.

- مرحلة الترجمة :

  في هذه المرحلة يتم تحويل البرنامج المصدر إلى برنامج بلغة الآلة .

2-4-5 **تنفيذ البرنامج**

يتم تجربة البرنامج لتأكد من البرنامج خالي من الأخطاء .

5-2 **شروط وخصائص الخوارزمية**

- المدخلات(input)

يجب ان تعرض القيم التي تحتاجها كمدخلات,صفر اواكثر من قيمة.

- المخرجات (output)

توضح الخوارزمية النتائج الفعلية المتوقعة من تطبيق الخوارزمية, قيمة أو أكثر.

- الوضوح(definiteness)

كل خطوة في الخوارزمية واضحة المعاني وغير غامضة.

- المحدودية(finiteness)

كل خطوات الخوارزمية يمكن حلها في فترة زمنية محددة.

- المحلولية (effectiveness)

كل خطوة في الخوارزمية تكون ممكنة الحل والفعالية.

- الاستثنائية :-

تعنى ان كل خطوة أو تعليمة من الخوارزمية لا تطابق مع الخطوات الآخر ى للخوارزمية.

6-2 طرق تمثيل الخوارزميات :

يتم تمثيل الخوارزميات بإحدى الطرق الآتية:-

1-6-2المخططات الانسيابية (flowchart) :

هي عب mارعن تمثيل رسومي للخوارزمية يوضح خطوات حل المشكلة من البداية إلى النهاية

مع إخفاء بعض التفاصيل لإعطاء صورة العامة للمشكلة.

تستخدم رموز معتمدة من المعهد الامريكى الوطني(a I n s I).

الجدول (2-1)التالي يوضح بعض الرموز والأشكال الهندسية المستخدمة.

الشكل	توضيح الشكل
START / END -1	رمز طرف المخطط (بداية أو نهاية) ويستعمل ليدل على بداية ونهاية مخطط سير العمليات.
INPUT / OUTPUT -2	رمز إدخال وإخراج يستعمل لإدخال البيانات أو لاستخراج النتائج.
PROCESSING-3	رمز المعالجة يستعمل للعمليات الحسابية ويكون في داخله معادله مثل : SUM=0 ، Z =X+Y

الشكل	توضيح الشكل
DECISION-4	رمز اتخاذ القرار يستعمل في حالات فحص قيمة معينة لاتخاذ قرار معين بالاعتماد على القيمة المفحوصة. ويكون مخرجاتها إما (YES , NO) وتكون كالآتي : No ؟ Yes
Connector-5	رمز التوصيل (الربط)
Flow Lines -6	خطوط التوصيل واتجاه السير

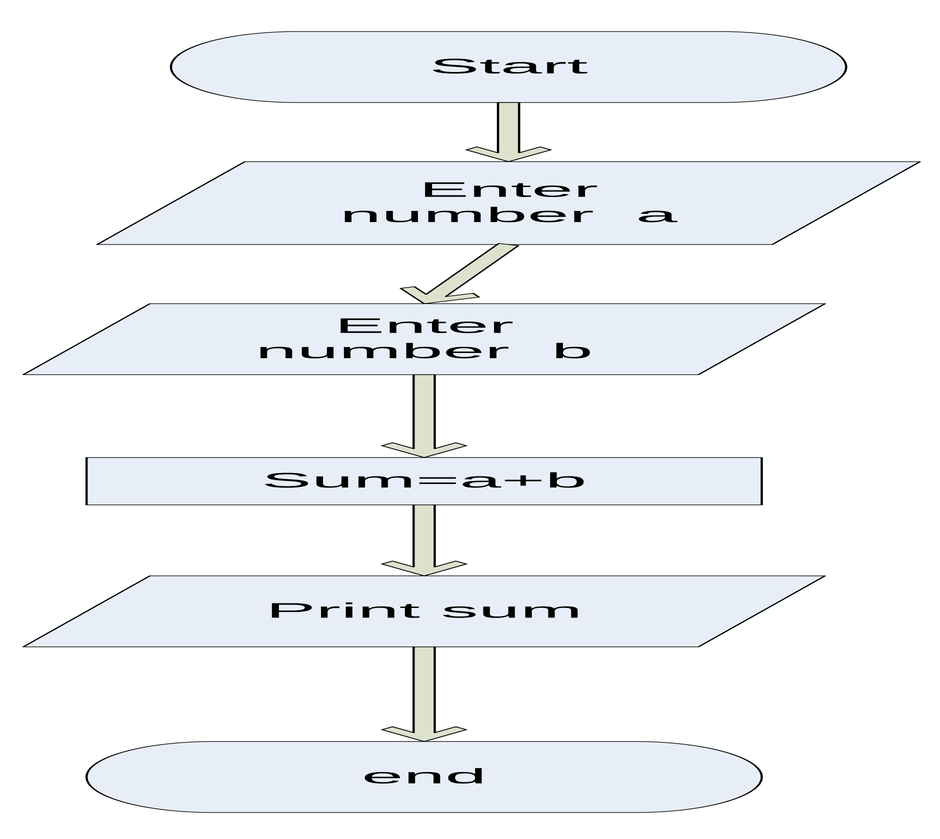
الجدول (2-1) الرموز والأشكال الهندسية المستخدمة لتمثيل الخوارزمية.

1-1-6-2 أنواع خرايط الانسيابية :

يمكن تصنيف خرائط سير العمليات إلى أربعة أنواع رئيسية وهي:

1. خرائط تدفق بسيطة

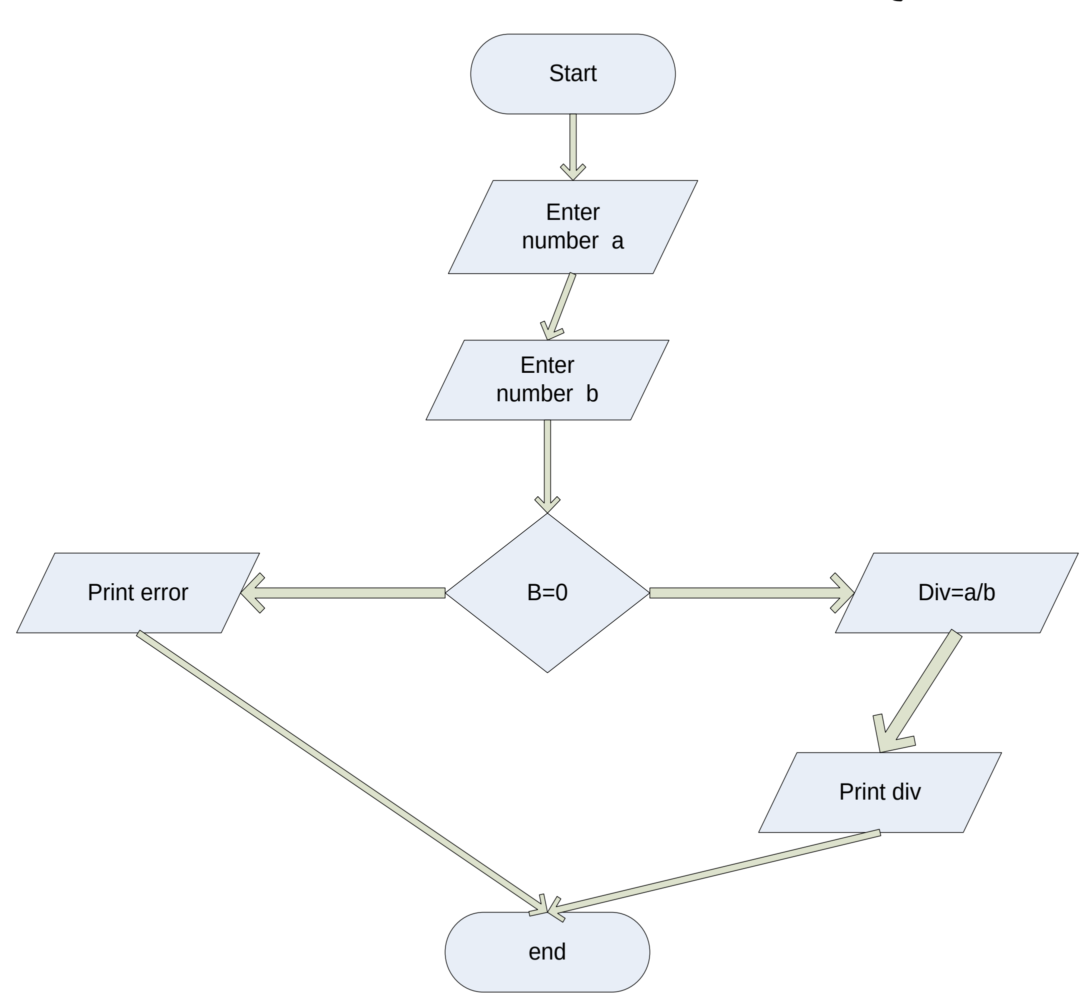
يكون في شكل تسلسل وتتبع بسيط خالي من التكرارات والاختبارات.



الشكل (2-1)يوضح خرائط تدفق بسيطة

2. مخططات تتدفق التفرع

يتضمن هذا النوع المفاضل بين نوعين من الحل باستخدام تكرار أو شرط معين.

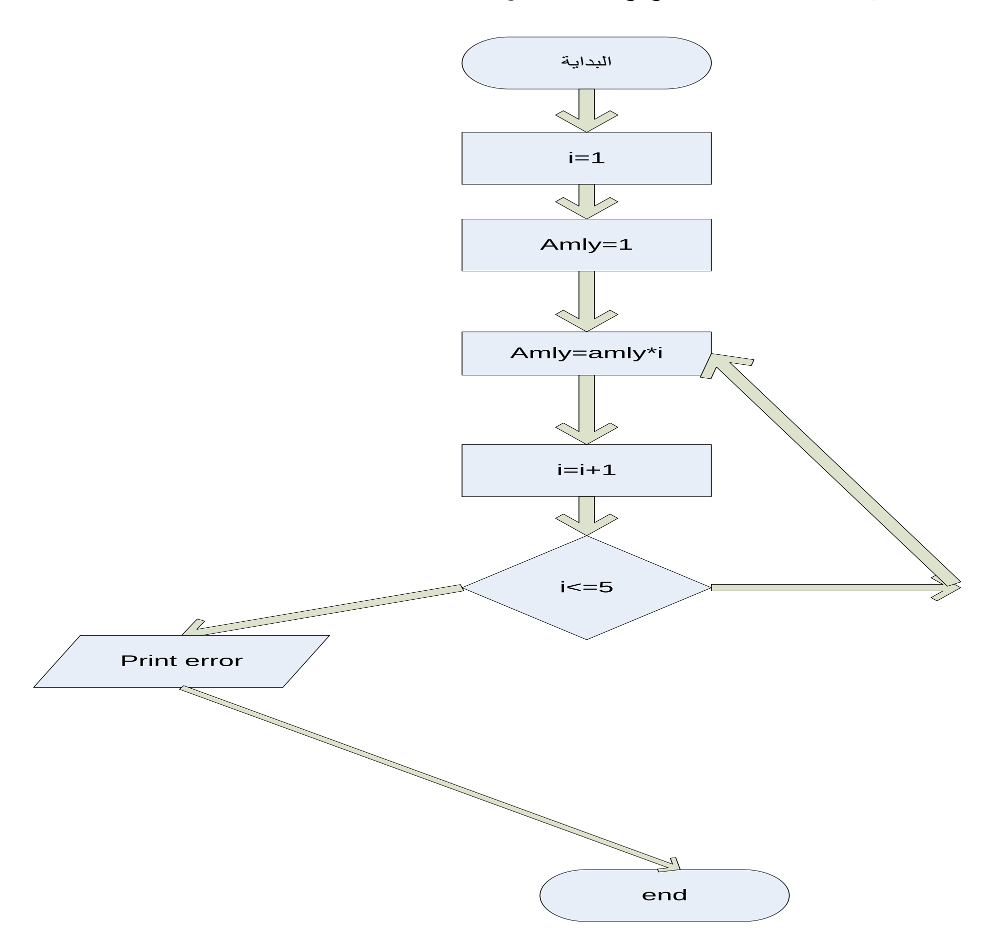


الشكل (2-2)يوضح مخططات تفرع التدفق

3. مخططات سير العمليات ذات التكرار

يستخدم هذا النوع من مخططات سير العمليات في الحالات التي تحتاج فيها إلى تكرار بعض العمليات عدد من المرات.

. يتم الخروج من التكرار في هذا النوع من المخططات باستخدام عداد يبدأ بقيمة، ويضاف إليه واحد في نهاية كل دورة، حتى تصل قيمة العداد عدد التكرارات المطلوبة.



الشكل (3-2)يوضح مخططات سير العمليات ذات التكرار

- تساعد المخططات التدفقية في تسهيل عملية دراسة البرنامج و مراجعته وتعديله واكتشاف أخطاء البرنامج.

- كذلك يستخدم كوسيلة لتوثيق البرامج بحيث يعكس المخطط كل العمليات الإدخال والإخراج ومعالجة البرنامج.

2-1-6-2 مساوئ المخططات الانسيابية

- تتميز المخططات بأنها صعبة نوعا ما وتحتاج إلى وقت كبير إلى لرسمها باليد,بالرغم من وجود حزم تطبيقية جاهزة عديدة ومتوفرة تسمح برسمها بالحاسوب.

-غير مستخدمة في البرامج الكبيرة والمعقدة,بعض الأحيان يصعب قراءتها وفهمها أمرين صعبين للغاية.

- تعديل المخططات ا صعب من تعديل شبة البرنامج وذلك لأنها تحتاج إعادة ترتيب ورسم

2-7-2 اللغة الطبيعة natural language

يتم تمثيل الخوارزمية بإحدى لغات البشر مثل اللغة العربية والانجليزية بطرق سهلة وبسيطة دون تعقيد.

لا يحتاج المستخدم إلى إتباع قوانين اللغة في كتابة الخوارزمية, تدعى اللغة الطبيعية باللغات المعتمدة على المعرفة (knowledge -based language) لأنة مرتبطة بموضوع معين.

تنتمي اللغة الطبيعية إلى لغات الجيل الخامس.

3-7-2 اللغة الرمزية

تمثيل الخوارزمية بلغات البشر كالانجليزية أو فرنسية أو العربية أو بلغات البرمجة كالباسكال (Pascal).البعض يستخدم الكثير من التفاصيل و البعض الآخر يستخدم القليل ... فلا قاعدة معينة لكتابة هذا النوع من الشفرات.

مثال لغة الرمزية

1- Start
2- Enter first number
3- Enter second number
4- If second number=0 then
a. Print error
b. Go to 6
5- Else
a. Calculate division
b. Print the result
6- end

7-2 أداء الخوارزمية Algorithm's Performance:

هنالك مقياسين أو أداتين مهمتين لقياس فعالية الخوارزمية هما :

1. تعقيدات الفراغ والخزن Space complexity.
2. تعقيدات الوقت Time complexity .

1-7-2 تعقيدات الفراغ والخزن Space complexity .

يقصد ب‍ه كمية المساحة المطلوبة بقرض تنفيذ الخوارزمية وهو عدد الخلايا الذاكرة التي استخدم ت بواسطة البرنامج ,ويعتمد على مايلى -:

أ. جزء ثابت

عبارة عن خصائص المدخلات المخرجات,ويتضمن الجزء فراغ التعليمات والمتغيرات والثوابت.

ب. الجزء متغير

يحجز لنظام التشغيل .

ويمكن صياغة التعقيد المكاني كما يلى:

CODE SEGMENT
DATA SEGMENT
HEAP SEGMENT
STACK SEGMENT

الجدول (2-2)يوضح التعقيد المكاني للخوارزمية

2-7-2 تعقيدات الوقت Time complexity:

هي كمية الوقت التي يتطلبها تشكيل البرنامج حتى اكتماله. ويتألف من الآني:

$$T(P) = CONST + TP$$

حيث أن

CONST : تمثل جزء خاص بوقت الترجمة.

TP: تمثل وقت تشغيل البرنامج.

8-2 أداء الخوارزمية Algorithm's Performance :

هناك خمس قواعد بموجبها نستطيع أن نختار الخوارزمية المناسبة لأداء العمل المطلوب وهى على النحو التالي :

1-8-2 صحة النتيجة:

لابد أن يكون النتاج هو الهدف الذي نصبو إليه, بمعنى انه لا تعتبر الخوارزمية صالحه لأداء العمل طالما النتيجة غير صحيحة . للتأكد من صحة الناتج لا يكفى أن نقارن بعض الامثل ة, فقد تكون النتيجة صحيحة لهذه الامثله ولكن عندما نضع معطيات أخرى تعطى نتيجة غير صحيحة.

الطريقة المثلى للتأكد من صحة النتيجة هي استخدام قواعد رياضيات للمعطيات والناتج, ومن ثم تطبيق هذه القواعد على الخوارزمية للتأكد من صحتها.

2-8-2 كمية العمل المطلوب:

كيف نقوم بقياس كميه العمل المطلوب لأداء الخوارزمية , استخدام الساعة هي الطريقة التي يعمد إليها الكثيرون ولكنها طريقه خاطئة لأنها تختلف باختلاف نوع وسرعة الحاسوب, كذالك نوع المعطيات يؤثر على الوقت المستغرق في أداء العمل.

لذالك لابد من أن نحلل الخوارزمية , والجزء الأهم في هذه الحالة هو الجز الذي يتكرر بعدد المعطيات ,أمثال loop, for و while وغيرها من الحلقات وما تحتويه من أوامر هي التي تحدد كميه العمل نظراً لأنها تتكرر عدد مرات أكثر كلما كبر حجم المعطيات.

3-8-2 الذاكرة المستخدمة:

أيضا في هذه الحالة يشرع الكثير من المبرمجين في تجربة الخوارزمية بمعطيات مختلفة ,ولكن كما ذكرنا في الحالة السابقة هذه ألطريقه خاطئة لأنها قد تنجح لبعض المعطيات ولكنها تفشل بمعطيات أخرى.

هنا نقوم بتحليل loop وغيرها من الحلزونيات التي تتكرر ,ونقارن المتغيرات وطريقه حفظها في الذاكرة , كما أن المعطيات تلعب دوراً كبيراً فلو فرضنا أن المعطيات هي مليون عدد, السؤال هو هل يمكن حفظ الإعداد في الذاكرة بطريقه أفضل ؟ هل يمكن ضغط المعطيات بحيث تأخذ حيز اقل؟

2-8-4 ألسهوله:

في العاده سهوله الخوارزمية شئ مطلوب, ولكن في بعض الأحيان قد تكون الخوارزمية السهلة ليست هي الفعالة , لذالك عند اختيار خوارزميه معينه لابد أن نضع في الاعتبار كثرة استخدامها , فإذا كانت ستستخدم بطريقه مستمرة قد يكون اختيار الخوارزمية الأكثر تعقيداً هو الاختيار الأنسب.

2-8-5 مثاليه:

كل خوارزميه تتطلب عدد من الخطوات التي لا بد منها, على سبيل المثال لترتيب العدد لابد أن تمر على كل عدد على الأقل مره واحده, وكذالك لابد من تغير مكان العدد التي توجد في غير موضعها الاصلى.

للوصول إلى المثالية في الخوارزمية , علينا أن نركز في التقليل من الخطوات, مع الأخذ في الاعتبار أن هناك خطوات لابد منها.

2-9 تحويل الخوارزمية إلى برنـامج:

يتم تحويل الخوارزمية بطريقه من اثنين, إما أن تكون الخوارزمية سهلة التحويل بحيث لا يتطلب من المبرمج سوى كتابه الشفرة المطلوبة , باى لغة كانت ,أو أن تكون الخوارزمية معقده وتتطلب من المبرمج اتخاذ قرارات معينه, مثلاً طريقه حفظ المعطيات, طريقه اختيار نوع المتغيرات, بحيث تتناسب معي اللغة التي يريد أن يستخدمها.

- ✓ من هذا نستخلص أن الخوارزمية لا علاقة لها بلغات البرمجة, وإنما تعتبر لغة برمجه معينه مجرد أداة لتطبيق الخوارزمية .

10-2 مقاييس تحليل الخوارزمية:

في بعض الأحيان يكون زمن تنفيذ الخوارزمية مختلف
هذا يكون بناء على عدد المدخلات حيث الحالة التي يكون فيها
زمن تنفيذ الخوارزمية قليل تسمى (best-case) , والحالة التي
تستغرق الخوارزمية زمن تنفيذ أطول بناء على عدد المدخلات
تسمى (worst-case) .

(best-case) و (worst-case) هما ليس تمثيلا للخوارزمية
, ولكن يفضل التحليل من خلال الـ (worst-case) لأنها توضح لنا
متى يمكن أن تكون الخوارزمية بطيئة, نجد هناك يضأ المتوسط
(average-case) حيث يتم حساب المتوسط لزمن التنفيذ خلال
عدد من المدخلات المتساوية وهو الحالة المثالية ولكن من الصعب
القيام به لأنه من الصعب تحديد الاحتمالات النسبية واختلاف
المدخلات لكثير من المشكلات.

11-2 أدوات تحليل الخوارزمية:

تستخدم مجموعة من الرموز notations لقياس كفاءة الخوارزمية(حيث نقصد بالكفاءة اقل وقت ممكن لتنفيذ الخوارزمية واقل سعة تخزينية) وتحليل أداءها كما تستخدم لحساب مدى تعقيد لخوارزمية . وهذه المقاييس هي:

Θ-Notation (Same order) 1-11-2

يستخدم داله مع عوامل ثابتة حيث نقول:

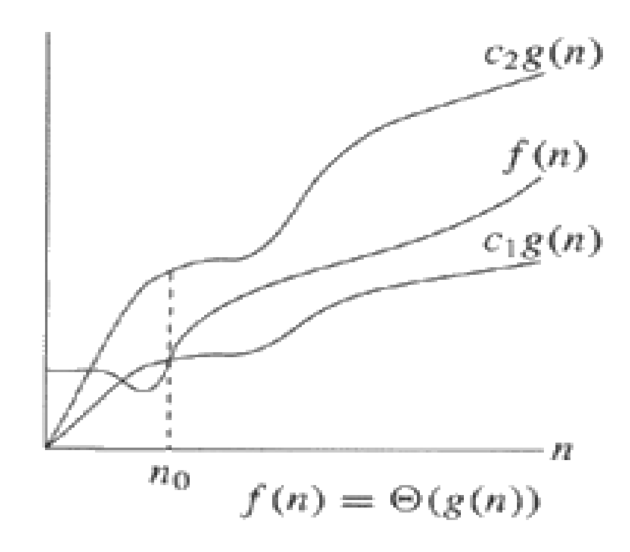
$$f(n) = \Theta(g(n))$$

إذا كانت هنالك ثوابت موجبة n0,c1 ,c2 حيث أنها تكون يمين n0, وقيمة f(n) دائما تقع بين c1g(n) و c2g(n) .

وتكتب كالآتي:

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_1, and n_0 such that

$$0 \le c_1\, g(n) \le f(n) \le c_2\, g(n) \text{ for all } n \ge n_0\}$$



شكل (4-2) تمثيل Θ-notation

O-Notation (Upper Bound) 2-11-2

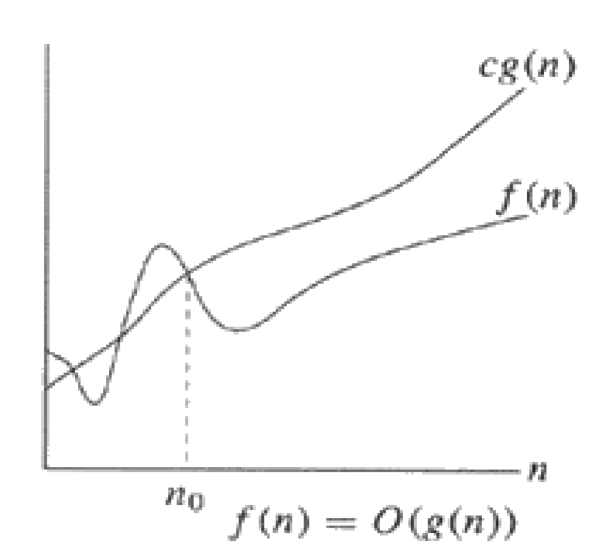
هذا المقياس يعطي الحد الأعلى للدالة في عوامل ثابتة.

$$f(n) = O(g(n))$$

وتكتب كالآتـي:

$O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that

$0 \le f(n) \le c\, g(n)$ for all $n \ge n_0\}$

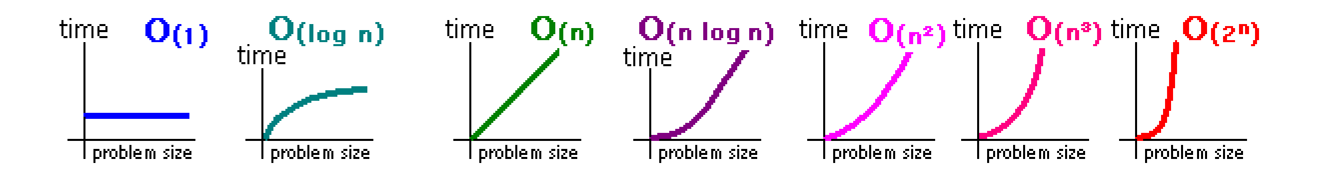


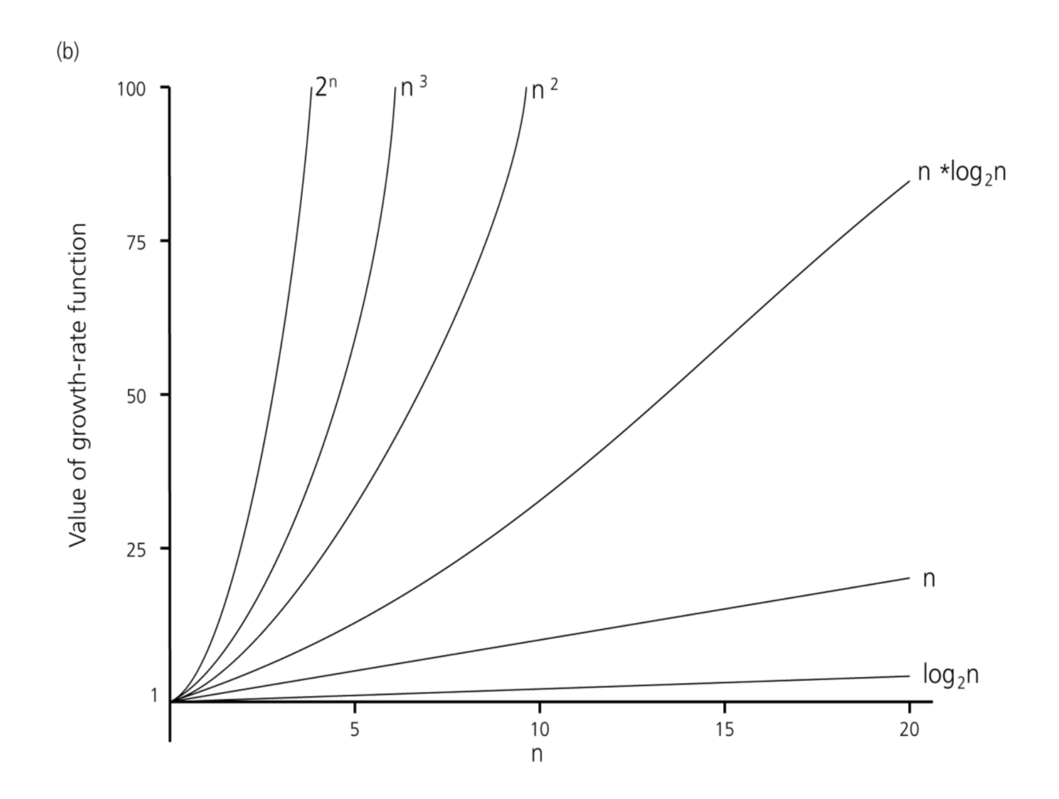
شكل (5-2) تمثيل **O-**Notation

1-2-11-2 بعض الدوال التي تقوم بوصف معدل النمو للخوارزمية :

1-constant	O(1)
2-logarithmic	O(log2 N)
3-linear	O(N)
4-n log n	O(N log2 N)
5-quadratic	O(N2)
6-cubic	O(N3)
7-exponential	$O(2^N)$

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$$





شكل (6-2) نمو الخوارزمية

2-2-11-2 مقارنة بين معدلات نمو الدوال Compare Growth rates functions

	n=1	n=2	n=4	n=8	n=16	n=32
1	1	1	1	1	1	1
logn	0	1	2	3	4	5
n	1	2	4	8	16	32
nlogn	0	2	8	24	64	160
n2	1	4	16	64	256	1024
n3	1	8	64	512	4096	32768
2n	2	4	16	256	65536	4294967296

جدول (2-3) مقارنة بين معدلات نمو الدوال

3-2-11-2 بعض الأمثلة لتحديد الـ Big O:

التكرار البسيط Simple Loops:

```
for  i: = 1 to  n do
k: = k + 5;
```

حلقة for يتم تحليلها كالاتى:

i: = 1 يتم تنفيذها مرة واحدة (أو وحدة من الزمن.) 1

i <= n يتم تنفيذها n+1 مرة

i++ يتم تنفيذها n مرة

إذن يتم تنفيذ كل الحلقة في (2n+2)

k: = k + 5; يتم تنفيذها n مرة

إذن مجموع زمن التنفيذ = 3n + 2

درجة التعقيد : (Complexity):

درجة التعقيد = O(n)

حيث يتم تجاهل الثوابت والمعاملات

الحلقات المتداخلة: Nested Loops

```
for  i: = 1 to  n do        ───────►   (2n+2)
for  j: = 1 to n do         ───────►   n(2n+2)
k := k + i + j;             ───────►   n2
```

مجموع زمن التنفيذ = 3n2+4n+2

درجة التعقيد (Complexity): = O(n2)

مثال آخر :

```
for  j: =1 to  n do         ───────►   (2n+2)
for  k:=1 to n do           ───────►   n(2n+2)
for   l:=1 to n do          ───────►   n2(2n+2)
sum:= sum+j * k * l;        ───────►   n3
```

مجموع زمن التنفيذ = 3n3+4n2+4n+2

درجة التعقيد (Complexity):

= O(n3)

دالة النداء الذاتي: Recursive Power function

```
Function  RecPow (var  X: double ; N:integer) : double
begin
if  N = 0
return 1                                   1
         else
return  X * RecPow(X, N – 1);              n-1
end;
```

درجة التعقيد =

$1 + (n-1) = n = \quad O(n)$

4-2-11-2 خصائص الـ Big O:

- تجاهل المعاملات أو الثوابت

If f(n) is c × g(n) then f(n) is O(g(n))

إذا كان لدينا الدالة التالية $f(n) = c \times g(n)$

فان $f(n) = O(g(n))$

حيث يتم تجاهل الثابت c

مثال:

$f(n) = 100*g(n) \longrightarrow f(n)$ is $O(n)$

- في حالة الضرب

إذا كان لدينا الدوال التالية

f1(n) is O(g1(n)) وكذلك f2(n) is O(g2(n)) فان

f1(n) × f2(n) is O(g1(n) × g2(n))

مثال:

f1(n) is O(n2)

f2(n) is O(n)

f1(n) × f2(n) is O(n2) × O(n) ⟶ O(n3)

هذه الخاصية تصلح في حالة الخوارزمية التي تحتوى على عدد من الحلقات التكرارية المتداخلة

- ❖ في حالة الجمع :

إذا كان لدينا الدوال التالية

f1(n) is O(g1(n)) وكذلك f2(n) is O(g2(n)) فان

f1(n) + f2(n) is O(g1(n)+ g2(n))

في هذه الحالة فان ال Big O تأخذ القيمة الأكبر إذن :

Big O = max (g1(n) +g2(n))

مثال (نفس المثال السابق)

```
for  j: =1 to  n do  ⟶  (2n+2)
for  k:=1 to n do    ⟶  n(2n+2)
for   l:=1 to n do   ⟶  n2(2n+2)
sum:= sum+j * k * l; ⟶  n3
```

مجموع زمن التنفيذ = 3n3+4n2+4n+2

درجة التعقيد (Complexity):

= O(n3)

5-2-11-2 محدودية الـ Big O : Limitations of Big-O

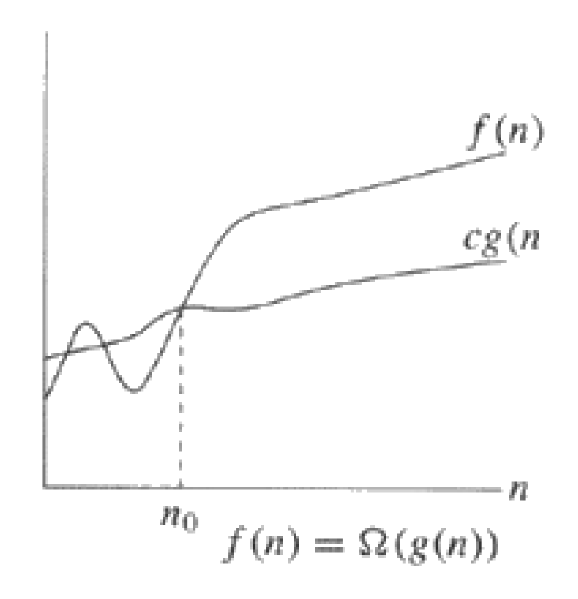
في بعض الأحيان تجاهل القيمة الثابتة ربما يؤدى إلى تغيير في القيم الناتجة

Ω-Notation (Lower Bound) 3-11-2

تمثل الحد الادني وتكتب كالآتي:

$$f(n) = \Omega(g(n))$$

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants c and } n_0 \text{ such that } 0 \le c\,g(n) \le f(n) \text{ for all } n \ge n_0\}$$



شكل (2-7) تمثيل **Ω-Notation**

12-2 العلاقة بين أدوات تحليل الخوارزميات :

Analysis Type نوع التحليل	Mathematical Expression التعبير الرياضى	Relative Rates of Growth معدل النمو
Big O	$T(N) = O(F(N))$	$T(N) < F(N)$
Big Ω	$T(N) = \Omega(F(N))$	$T(N) > F(N)$
Big θ	$T(N) = \theta(F(N))$	$T(N) = F(N)$

جدول (2-4) العلاقة بين ادوات تحليل الخوارزميات

13-2 معدل نمو الخوارزمية T(N)

F(N) هى الدالة التي تحدد معدل النمو ربما تكون قيمة عظمى أو قيمة صغرى may be upper or lower bound

ربما تكون مساوية للـ T(N)

على الرغم من الدقة الاضافية التي تقدمها الـ Big Theta الا أن الـ Big O هى الاكثر شيوعا واستخداما الا عند عدد قليل من الباحثين في مجال تحليل الخوارزميات مثل Mark Weiss

14-2 تحليل بعض الخوارزميات باستخدام الـ Big O :

1-14-2 خوارزمية الفقاعة: Bubble sort

```
for  i:=n-1 down to 1 do  ──────►        (2n+2)
for  j:=1 to i do         ──────►    n(2n+2)
if arr[j]>arr[j+1] then   ──────►        n2
begin
swap:=arr[j];
arr[j]:=arr[j+1];
arr[j+1]:=swap;
end;
```

درجة التعقيد = O(n2)

2-14-2 الترتيب بالاختيار:Selection Sort

```
for  h:=1 to n do              ──────►      (2n+2)
begin
min:=h;                        ──────►      n
for  m:=h+1to n do             ──────►      n(2n+2)
if arr[m]<arr[min] then        ──────►      n
min:=m;
temp:=arr[h];
arr[h]:=arr[min];
arr[min]:=temp;
```

```
end;
```

O(n2)= درجة التعقيد

3-14-2 الترتيب السريع:Quick sort

```
procedure  Qsort(var arr:art;first, last:integer);
var
m, P,L,R,T:integer;
 begin
L:=first; R:=last; M:=(first+last) div 2+1;
P:=arr[M];
while(L<=R) do              n
begin
while(arr[L] < P)  do       n(n/2)
L:=l+1;                   (n-1)
while(arr[R]>P) do   n(n/2)
R:=r-1      ;             n-1
if L<=R then              n
begin
T:=arr[L]; arr[L]:=arr[R]; arr[R]:=T; L:=L+1  ;   R:=R-1;
end; end;
 if first<R  then    n
Qsort(arr,first,R);   n(n-1)
```

```
if L<last then
Qsort(arr,L,last);    n(n-1)
End;
```

O(n log n)= درجة التعقيد

3-1 مقدمة:

يعتبر ترتيب المعلومات من الاستخدامات المتكررة في معالجة البيانات، ونعنى بترتيب المعلومات وضع السجلات حسب ترتيب معين ويتم ذلك بالطبع عن طريق ترتيب مفاتيح السجلات التي ترتب ترتيباً تصاعدياً أو تنازلياً. فإذا كان المفتاح مفتاحاً رقمياً مثل رقم الطالب أو رقم الموظف أو رقم البطاقة الشخصية فإن ترتيب المفاتيح يتم من أصغر رقم أي أصغر قيمة مفتاح تصاعدياً إلى أكبر رقم، أي أكبر قيمة مفتاح، وبهذا تكون سجلات الطلاب أو الموظفين أو المواطنين تم ترتيبها تصاعدياً، والعكس، إذا بدأنا الترتيب بأكبر رقم ثم نزلنا لأصغر يكون الترتيب تنازلياً، وبالمثل إذا كان المفتاح بالأسماء فإن الترتيب يكون أبجدياً مبتدئاً بالحرف الأول للاسم بالألف ومنتهياً بالياء وعندما يتساوى مفتاحان في الحرف الأول ينظر إلى الحرف الثاني وإذا تساوى في الحرف الثاني ينظر إلى الحرف الثالث وهكذا، من الضروري ألا يتطابق اسمان في كل الحروف و إلا يكون حقل الاسم غير صالحٍ ليكون مفتاحاً، لأن هنالك شرطاً أساسياً لمفتاح السجل وهو أن يكون مفرداً في التعبير عن السجل (لهذه المشكلة لا يفضل استخدام الأسماء في المفاتيح).

إن أهم استخدامات الترتيب هو تسهيل أو تسريع عملية البحث عن السجل هذا إضافة إلى استخدامات أخرى في ترتيب المستويات للأفراد مثل نتائج الامتحانات والمنافسات بين الطلاب والموظفين وعامة المتنافسين ومثل أولويات الحجوزات والطلبات والخدمات عموماً والتي تعطي أولوية للأول فالتالي وهكذا، إذن ليس كل الترتيب يتم فقط لحقل المفاتيح وإنما يمكن أن يتم على أي حقل حسب الحاجة الاستفسارية أو التحليلية.

2-3 تعريف الترتيب Sorting :

الترتيب هو عملية تنظيم مجموعة من العناصر البيانية وفق قيمة حقل (او مجموعة حقول) يسمى بالمفتاح بصورة تصاعدية او تنازلية.

- الغرض من الترتيب هو-:
  1. لزيادة كفاءة خوارزميات البحث عن عنصر ما .
  2. لتبسيط معالجة الملفات
  3. لحل مشكلة تشابه القيود
- يوجد عدة تصنيفات لخوارزميات الترتيب منها-:
  1. خوارزميات تعتمد المقارنات
  2. خوارزميات تعتمد التوزيع
  3. خوارزميات تعتمد على اجهزة الخزن وتتضمن-:

     أ. خوارزميات الترتيب الداخلي

     ب. خوارزميات الترتيب الداخلي

3-3 خطوات عملية الترتيب:

خوارزميات الترتيب بمختلف انواعها تتضمن الخطوات التالية

1. قراءة حقل المفتاح
2. استنتاج موقع العنصر في الترتيب الجديد
3. نقل العنصر إلى الموقع الجديد

4-3 اسس قياس كفاءة خوارزميات الترتيب :

1. معدل ما تحتاجه الخوارزمية من مقارنات
2. معدل ما تحتاجه الخوارزمية من النقلات او التحريكات
3. معدل ما تحتاجه الخوارزمية من التبديلات
4. معدل الحجم التخزيني

- تنقسم خوارزميات الترتيب بصورة عانة إلى قسمين هما :
  1. خوارزميات الترتيب الداخلي Internal Sort Algorithm
  2. خوارزميات الترتيب الخارجي External Sort Algorithm

3 5 خوارزميات الترتيب الداخلي Internal Sort Algorithms

إذا كان الملف المراد ترتيب بياناته معبأ ببيانات داخل الذاكرة ,كأن يتم تعبئة البيانات في صورة مصفوفة داخل الذاكرة .في هذه الحالة يتم تريب البيانات ترتيبا داخليا Internal sort ,وفي هذه الحالة يمكن الوصول إلى اي سجل بطريقة سهلة .

وينقسم إلى:

3 5 1 طرق الترتيب باستخدام التبديلات :

وتتضمن عدة خوارزميات هي:

i. خوارزمية الترتيب الفقاعي
ii. خوارزمية الترتيب السريع
iii. خوارزمية شيل
iv. خوارزمية شيكر
v. خوارزمية الازاحة
vi. خوارزمية باتجر
vii. خوارزمية الترتيب الفردي والزوجي

3 5 2 طرق الترتيب بالاضافة :

وتتضمن عدة خوارزميات هي :

i. الترتيب بالاضافة إلى القائمة
ii. الترتيب بحساب العنوان
iii. الترتيب بالاضافة الثنائي
iv. الترتيب بالاضافة الخطي

3-5-3 طرق الترتيب باستخدام الدمج

وتتضمن عدة خوارزميات هي:

i. خوارزمية الدمج البسيط
ii. خوارزمية الدمج المستقيم
iii. خوارزمية الدمج الطبيعي

4-5-3 طرق الترتيب التوزيعي:

وتتضمن خوارزمية واحدة فقط هي-:

i. خوارزمية ترتيب الاساس .

5-5-3 طرق الترتيب بواسطة الاختيار

وتتضمن عدة خوارزميات هي :

i. خوارزمية الاختيار الخطي
ii. خوارزمية الاختيار التربيعي
iii. خوارزمية الترتيب الكومي
iv. خوارزمية الترتيب الشجري
v. خوارزمية الاختيار الخطي بالتبديلات

3 6 ثبات خوارزمية الترتيب Stability of sorting algorithm :

تكون الخوارزمية ثابتة إذا كان يحافظ على الترتيب النسبي للكميات المتساوية بالنسبة لعلاقة الترتيب.

مثال, بالنسبة للعناصر الآتية:

(6 , 5) (7, 3) (1 , 3) (1 , 4)

الذي نرتبها حسب الاحداثية الأولى (المفتاح) نجد حالتين, عندما يتم احترام الترتيب النسبي وعندما لا يحترم :

(1 , 3) (7, 3) (1, 4) (6 , 5) (ترتيب نسبي محترم).

(7, 3) ((1 , 3) ((1, 4) ((6 , 5)) (ترتيب نسبي متغير).

3-7 تصنيف خوارزميات التصنيف عبر خصائص الخوارزميات :

هنالك تصنيفان أساسيان لخوارزميات الترتيب هما

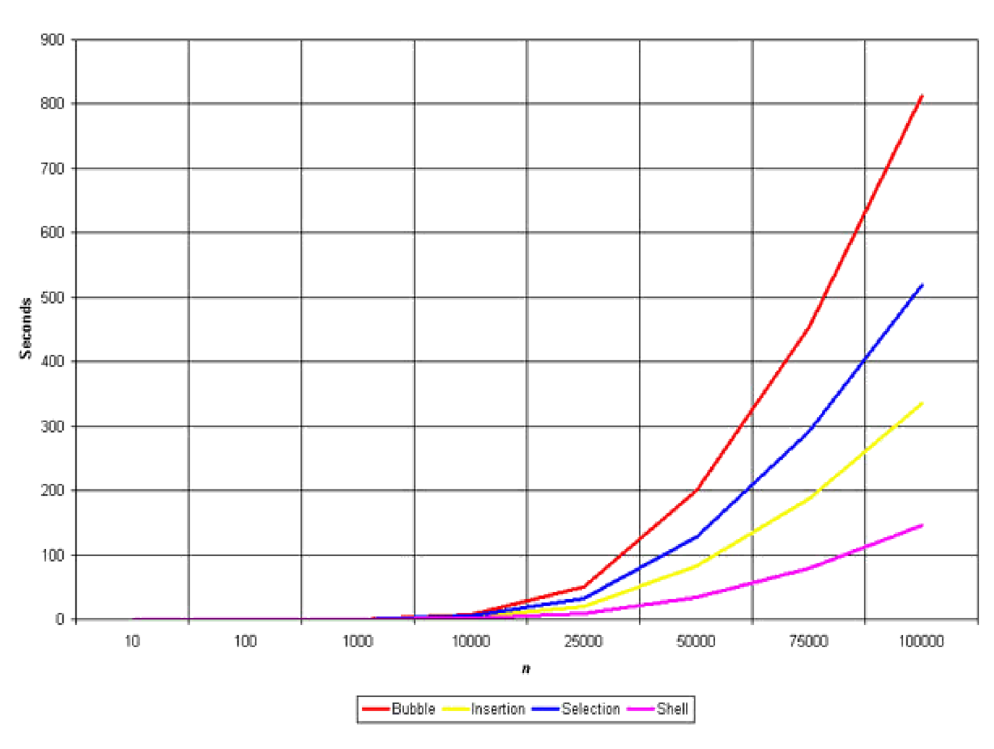
- $O(n^2)$-algorithms.

ويكون زمن تنفيذ هذه الخوارزميات هو:

$$T(n)= O\ (n^2)$$

وتشمل خوارزميات :

- Bubble sort.
- Insertion sort.
- Selection sort.
- Shell sort.



شكل (3-1) خوارزميات $O(n^2)$

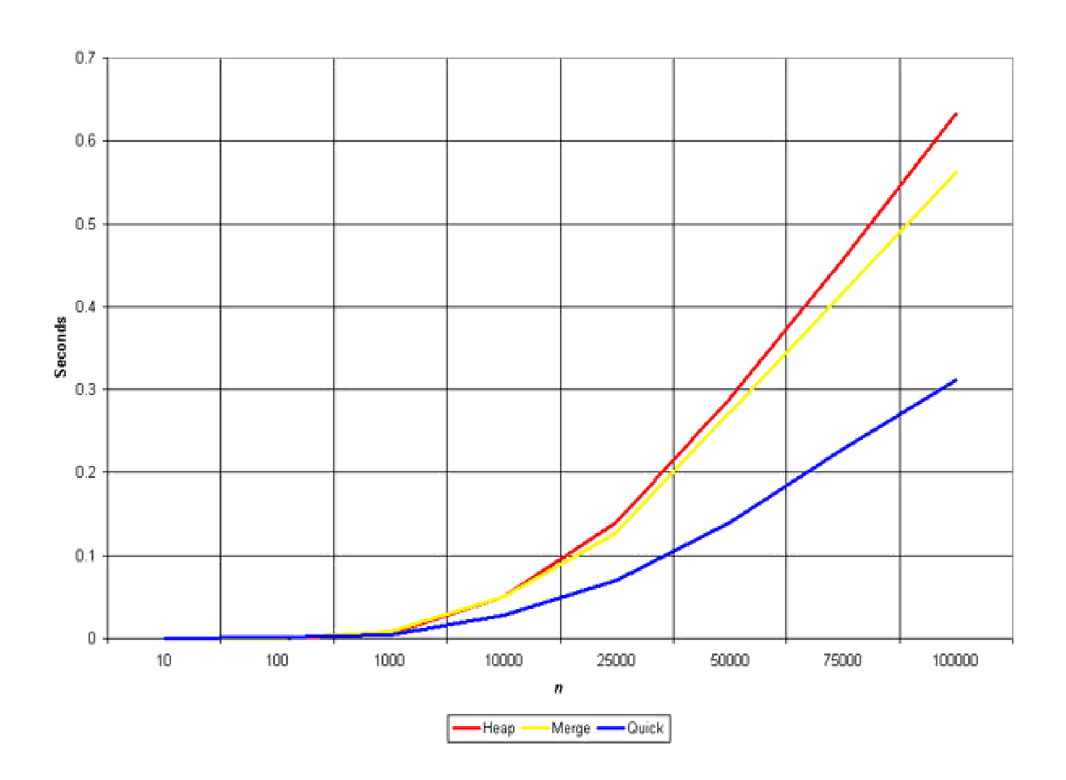
- O(*n log n*)-algorithms.

ويكون زمن التنفيذ في هذه الخوارزميات هو

$$T(n)= O(n \log n)$$

وتشمل خوارزميات :

  - heap sort.
  - merge sort.
  - quick sort.



شكل (2-3) خوارزميات -O(*n log n*)

Shell sort algorithm 8 3

1-8-3 تعريف:

هي امتداد لخوارزمية الترتيب بالإدخال insertion sort وهي أسرع من خوارزمية الترتيب بالإدخال حيث تقوم بتبديل عناصر متباعدة عن بعضها البعض (تقوم خوارزمية الترتيب بالإدخال بتبديل العناصر المتجاورة فقط).

صممت خوارزمية شل على أساس خوارزمية الترتيب بالإدخال وأكثر كفاءة منها حيث أن خوارزمية الترتيب بالإدخال تكون كفوءة إذا كان قليلة (قائمة قصيرة) أو إذا كانت العناصر مرتبة مسبقا في القوائم الطويلة.

تقوم خوارزمية الشل على أساس تقسيم القائمة إلى أجزاء قصيرة وترتيب كل جز ء بواسطة مفهوم الترتيب بالإدخال ثم تقوم بترتيب كل القائمة من خلال الأجزاء المرتبة مسبقا.

إذن تزيد كفاءة الخوارزمية كلما كانت القائمة طويلة. حيث تعتبر تقريبا 5 مرات أسرع من خوارزمية الفقاعة bubble sortومرتين أسرع من خوارزمية الترتيب بالإدخال insertion sort.

تقوم فكرة الخوارزمية على أساس ترتيب عناصر الملف من خلال أخذ عدد h^{th} عنصر (مبتدئً من اي مكان) .

مثال:

إذا أردنا ترتيب القائمة التالية بأخذ 3 عناصر في كل مرة .

34	21	40	12	27	18	29	13	25	17	11	38

نقوم أولا بأخذ العناصر في المواقع 0, 3, 6, 9 ثم العناصر في المواقع 1, 4, 7, 10 ثم العناصر في المواقع 2, 5, 8, 11

كالأتي:

34	12	29	17
21	27	13	11
40	18	25	38

12	17	29	34
11	13	21	27
18	25	38	40

ثم نقوم بجمعهما مرة أخري ليصبح شكل القائمة كالأتي:

12	11	18	17	13	25	29	21	38	34	27	40

إذا أجرينا مقارنة بين القائمة الأصلية (غير المرتبة) والقائمة بعد إجراء الترتيب بأخذ 3 عناصر 3-sorted مع القائمة المرتبة نلاحظ إن العديد من العناصر قطعت مسافة طويلة لتصل إلى مكانها .

Unsorted:	34	21	40	12	27	18	29	13	25	17	11	38
3-Sorted:	12	11	18	17	13	25	29	21	38	34	27	40
Sorted:	11	12	13	17	18	21	25	27	29	34	38	40

إذا نلاحظ أن خوارزمية شل تقوم على فكرة تكرار ترتيب قوائم فرعية من القائمة الرئيسية باستخدام مفهوم الترتيب بالإدخال وذلك بأخذ h^{th} عنصر في كل مرة ولزيادة كفاءة الخوارزمية نقوم باختيار h كبير نسبيا ومن ثم في المرة التالية نقوم بإنقاص قيمة h حتى تصبح h=1 لتصبح القائمة مرتبة بصورة كاملة.

مثال آخر:

لنجري الترتيب على القائمة التالية بأخذ 7 عناصر في المرة الأولى h=7 ومن ثم في الخطوة الثانية أخذ 3 عناصر h=3 والمرة الثالثة عنصر واحد h=1:

كما في الشكل التالي:

Unsorted	64	31	10	40	22	49	82	20	29	56	40	18	19	27	26
7-sorted	20	29	10	40	18	19	27	26	31	56	40	22	49	82	64
3-sorted	20	18	10	27	26	19	40	29	22	49	40	31	56	82	64
1-sorted	10	18	19	20	22	26	27	29	31	40	40	49	56	64	82

First Pass:		
20	26	64
29	31	
10	56	
40	40	
18	22	
19	49	
27	82	

Second Pass:				
20	27	40	49	56
18	26	29	40	82
10	19	22	31	64

لكن السؤال ما هو انسب قيم لـ h في كل مرة؟

الإجابة على هذا السؤال حتى الآن غير معروفة لكن الأنسب بأخذ قيمة h عبر المعادلة التالية:

$$h = h/3 + 1$$

هذه المعادلة تعطينا أرقاما تبدو كالأتي:

..., 1093, 364, 121, 40, 13, 4, 1

2-8-3 وصف الخوارزمية:

```
SHELL_SORT (A)
for h = 1 to h ≤ N/9 do
   for (; h > 0; h != 3) do
      for I = h +1 to I ≤ n do
         v = A[I]
         j = I
         while (j > h AND A[j - h] > v
               A[I] = A[j - h]
               j = j - h
         A[j] = v
         I = I + 1
```

3-8-3 خطوات الخوارزمية :

تقوم بتقسيم القائمة المراد ترتيبها إلى مجموعة من المسافات الوهمية.

ثم يتم المقارنة بين بين عنصرين أو أكثر غير متجاورين وإنما متباعدين بمسافة محددة.

ثم يتم اختصار المسافة الوهمية إلى النصف وتجرى المقارنة مرة أخرى لعنصرين وتختصر المسافة إلى أن تصبح المسافة بين العنصرين واحد وبذلك تكون القائمة قد تم ترتيبها.

خصائص خوارزمية شيلا :

تزداد كفاءتها كلما ازدادت عدد القيود .

لا تحتاج إلى مكان إضافي في الذاكرة.

4-8-3 إجراء الخوارزمية:

```
void shellSort(int numbers[], int array_size)
{
  int I, j, increment, temp;
  increment = 3;
  while (increment > 0)
  {
    for (I=0; I < array_size; I++)
    {
      j = I;
      temp = numbers[I];
      while ((j >= increment) && (numbers[j-increment] > temp))
      {
        numbers[j] = numbers[j - increment];
        j = j - increment;
```

```
      }
      numbers[j] = temp;
    }
    if (increment/2 != 0)
      increment = increment/2;
    else if (increment == 1)
      increment = 0;
    else
      increment = 1;
  }
}
```

: Merge sort algorithm 9 3

1-9-3 مفهوم خوارزمية الترتيب بالدمج

خوارزمية الترتيب بالدمج merge sort algorithm تقوم على مفهوم خوارزميات التقسيم والضم divide-and-conquer .

يعتبر worst-case running time اقل من خوارزميات الترتيب بالإدخال. تقوم بتقسيم القائمة إلى جزأين (قائمتين فرعيتين) A[p...r], حيث إن p=0 و r=n في البداية لكن هذه القيم تتغير في الخطوات التالية.

2-9-3 خطوات الخوارزمية:

وتتكون الخوارزمية من ثلاث خطوات أساسية هي:

3-9-2-1 Divide Step:

إذا كان لدينا القائمة A المراد ترتيبها فإذا كان عدد العناصر بهذه القائمة هو صفر أو واحد هذا يعني إن القائمة مرتبة ولا داعي لإجراء اى ترتيب. خلاف ذلك قم بتجزئة القائمة A[p…r] إلى جزأين (قائمتين فرعيتين) A[r..q] و A[q+1…r] كل قائمة تحتوى تقريبا على نصف عناصر القائمة الأصلية والنقطة qهي تقريبا النقطة الوسطى في القائمة A[p…r] .

3-9-2-2 Conquer Step

في هذه المرحلة يتم ترتيب كل قائمة فرعية على حدا .وذلك بتكرار الخطوة الأولى عدة مرات.

3-9-2-3 Combine Step:

في هذه المرحلة يتم جمع القوائم الفرعية المرتبة مرة أخرى لتكوين القائمة الرئيسية.

مع ملاحظة إن هذه المرحلة يتم استدعاءها بعد تقسيم القوائم الفرعية إلى قوائم فرعية اصغر حتى يصبح طول القائمة عنصر واحد.

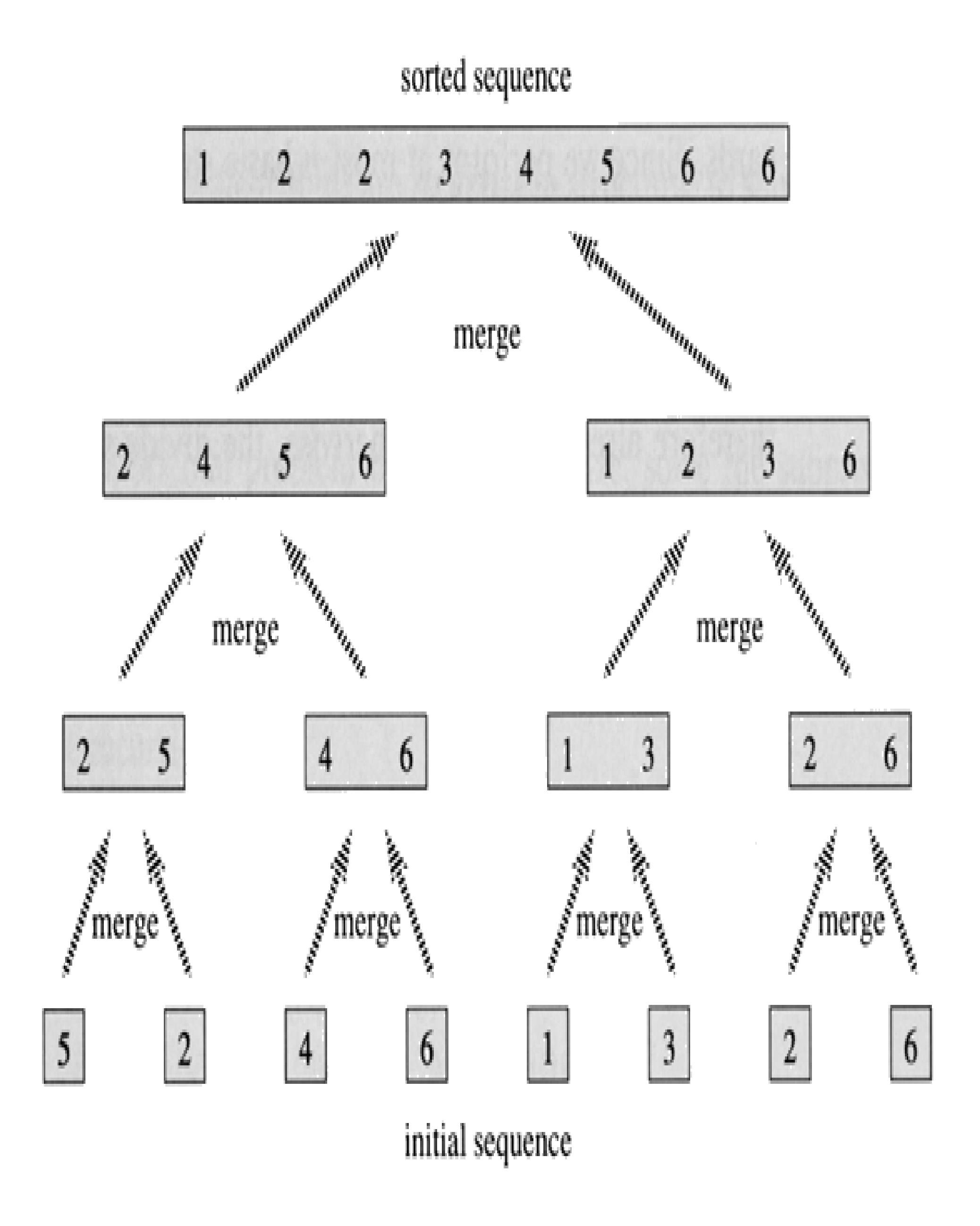
3-9 3 وصف الخوارزمية:

```
MERGE-SORT (A, p, r)

1.   IF p < r                         // Check for base case
2.   THEN q = FLOOR[(p + r)/2]   // Divide step
3.   MERGE (A, p, q)                  // Conquer step.
4.   MERGE (A, q + 1, r)          // Conquer step.
5.   MERGE (A, p, q, r)           // Conquer step.
```

مثال:

إذا كان لدينا قائمة فرعية طولها 8 عناصر وأردنا ترتيبها باستخدام خوارزمية الدمج كما مبين في الأسفل. (الدمج من أسفل إلى اعلي)



4-9-3 الخوارزمية :

INPUT: Array A and indices p, q, r such that $p \leq q \leq r$ and subarray A[p .. q] is sorted and subarray A[q + 1 .. r] is sorted. By restrictions on p, q, r, neither subarray is empty.

OUTPUT: The two subarrays are merged into a single sorted subarray in A[p .. r].

The pseudocode of the MERGE procedure is as follow:

```
MERGE (A, p, q, r )

1.    n1 ← q − p + 1
2.    n2 ← r − q
3.    Create arrays L[1 . . n1 + 1] and R[1 . . n2 + 1]
4.    FOR I ← 1 TO n1
5.        DO L[I] ← A[p + I − 1]
6.    FOR j ← 1 TO n2
7.        DO R[j] ← A[q + j ]
8.    L[n1 + 1] ← ∞
9.    R[n2 + 1] ← ∞
10.   I ← 1
11.   j ← 1
12.   FOR k ← p TO r
13.       DO IF L[I ] ≤ R[ j]
14.            THEN A[k] ← L[I]
15.                 I ← I + 1
```

```
16.            ELSE A[k] ← R[j]
17.                 j ← j + 1
```

5-9-3 تحليل الخوارزمية:

1. Divide:

في هذه المرحلة يتم فقط حساب قيمة نقطة الوسط q وبالتالي تاخذ زمن تنفيذ ثابت Θ(1)

2. Conquer:

في هذه المرحلة يتم معالجة القائمتين الفرعيتين عبر تكرار ذاتي طول كل قائمة هو n/2 إذن زمن التنفيذ هو 2T(n/2) .

3. Combine:

في هذه المرحلة يتم جمع العناصر n في قائمة واحدة إذن زمن تنفيذها هو Θ(n)

إذن بجمعها مع بعضها يصبح زمن تنفيذ الخوارزمية هو:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

6-9-3 أجراء خوارزمية الترتيب بالدمج:

```
void mergeSort(int numbers[], int temp[], int array_size)

{
    m_sort(numbers, temp, 0, array_size - 1);


}
void m_sort(int numbers[], int temp[], int left, int right)


{
    int mid;
    if (right > left)
   {

        mid = (right + left) / 2;

        m_sort(numbers, temp, left, mid);

        m_sort(numbers, temp, mid+1, right);

        merge(numbers, temp, left, mid+1, right);
    }

}
```

```
void merge(int numbers[], int temp[], int left, int mid, int
right)

    {
        int I, left_end, num_elements, tmp_pos;

        left_end = mid - 1;
        tmp_pos = left;
        num_elements = right - left + 1;

while ((left <= left_end) && (mid <= right))
    {
        if (numbers[left] <= numbers[mid])
        {
            temp[tmp_pos] = numbers[left];
            tmp_pos = tmp_pos + 1;
            left = left +1;
        }
        else
        {
            temp[tmp_pos] = numbers[mid];
            tmp_pos = tmp_pos + 1;
            mid = mid + 1;
        }
    }
```

```
while (left <= left_end)
   {
            temp[tmp_pos] = numbers[left];
            left = left + 1;
            tmp_pos = tmp_pos + 1;
      }
      while (mid <= right)
      {
            temp[tmp_pos] = numbers[mid];
            mid = mid + 1;
            tmp_pos = tmp_pos + 1;
      }

      for (I = 0; I <= num_elements; I++)
      {
            numbers[right] = temp[right];
            right = right - 1;
      }
}
```

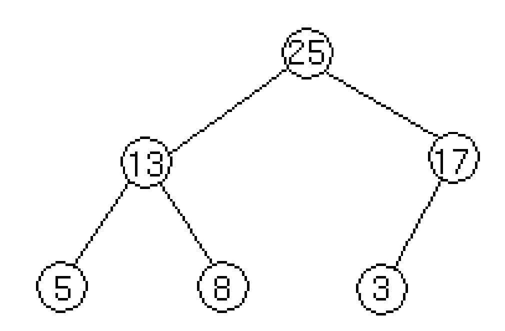
: Heap Sort Algorithm   10 3

3-10-1 مفهوم الخوارزمية :

تستخدم خوارزمية الكومة heap sort algorithm بنية بيانات عبارة عن شجرة ثنائية binary tree .

يتم استخدام المصفوفات لتخزين البيانات بحيث يمكن عرضها في صورة شجرة ثنائية كاملة complete binary tree. كل عقدة في الشجرة الثنائية متوافقة مع عنصر في المصفوفة .

تكون المصفوفة معبئة بصورة كاملة بالعناصر ما عدا المستوى الأخير من الممكن إن يكون ناقصا .



شكل (3-3) شجرة ثنائية معبئة ما عدا المستوى الأخير

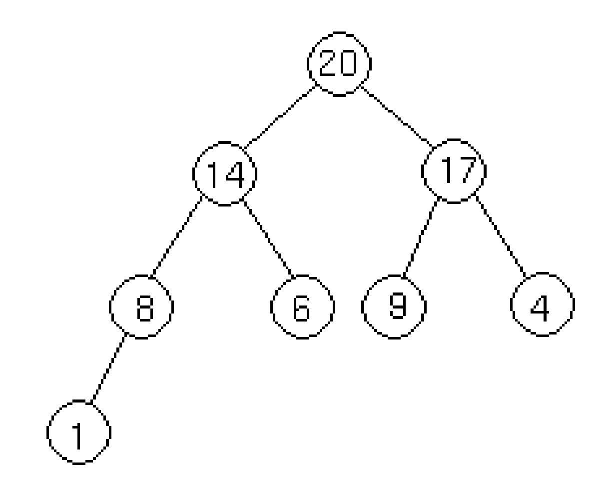
يتم تمثيل الكومة heap في مستويات مرتبة . انطلاقاً من اليسار إلى اليمين .إذا المصفوفة للشجرة الثنائية السابقة هي:

[25 , 13 ,17 , 5 , 8 , 3]

جزر الشجرة root هو A[1] يؤشر إلى العقدة I .مؤشرات العقدة الأب والابن الأيسر والابن الأيمن يمكن إن تحسب.

```
PARENT (I)
    return floor(I/2
LEFT (I)
    return 2i
RIGHT (I)
    return 2i + 1
```

دعنا نحاول إثبات والتأكد من الشجرة صحيحة للكومة heap التالية:



شكل (3-4) تمثيل الكومة في شجرة ثنائية

المصفوفة للشجرة السابقة هي :.[20, 14, 17, 8, 6, 9, 4, 1]

- إذا أردنا الانطلاق من العنصر 20 إلى العنصر 6:
  - مؤشر (index) العنصر 20 هو 1.

- لإيجاد مؤشر العنصر الابن الأيسر يمكن حسابه من خلال المعادلة 1*2=2 .ويعطينا (بصورة صحيحة) العنصر 14 .
- لإيجاد مؤشر العنصر الأيمن للعنصر 14 يتم حسابه من خلال المعادلة 2*2+1=5 وهو يعطينا العنصر 6 بصورة صحيحة تماما.

- إذا أردنا الانطلاق من العنصر 4 إلى العنصر 20 :
  - مؤشر العنصر 4 هو 7 .
  - لإيجاد مؤشر العنصر الأب يمكن حسابه بالمعادلة 7\2=3 ويعطينا العنصر 17.
  - لإيجاد مؤشر العنصر للعنصر 17 يمكن حسابه بالمعادلة 3\2=1 ويعطينا العنصر 20 وهو المطلوب بصورة صحيحة.

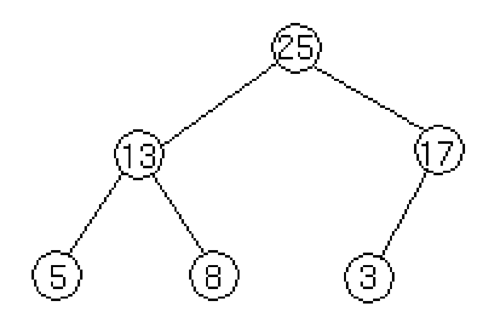
2-10-3 خصائص الكومة : heap

لكل عقدة في I غير العقدة الجزر, قيمة العقدة اقل من أو تساوي (على الأكثر) العقدة قيمة العقدة الأب لها .

$$A[PARENT(I)] \geq A[I]$$

وهذا يؤدي بالضرورة إلى إن اكبر عقدة هي العقدة الجزر root .

مثلا:

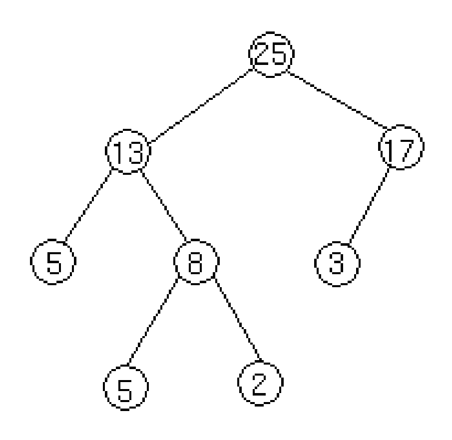


شكل (3-5) مثال شجرة ثنائية

من خلال التعريف , كل مستويات الشجرة يجب إن تملأ ما عدا المستوى الأخير .بحيث تملأ من الشمال لليمين. ارتفاع الكومة h هو اصغر عنصر في الكومة ويكون عبارة عن عقدة واحدة فقط موجودة في أسفل مستوى في الكومة , المستوى الأعلى مباشرة من أسفل مستوى في الشجرة الثنائية هو المستوى h-1 وبه 2^h -1 عقدة.

إذن اصغر عنصرا في الشجرة من الممكن أن يكون في العقدة 2^h في حالة إن المستوى الأخير (اقل مستوى) تمت تعبئته بالكامل فان عدد عقد الشجرة من خلال ارتفاعها هو 2^{h+1} -1 عقدة .

في المثال التالي هذه الشجرة ليست كومة not heap , لأنها ليست شجرة ثنائية لان العقد في المستويات لم يتم ملأها جميعها . ومن خلال خصائص الكومة يجب إن تملأ جميع المستويات ما عدا المستوى الأخير يمكن إن يكون ناقصا وذلك يملأها من الشمال إلى اليمين :



شكل (3-6)تعبئة الشجرة الكومة في الشجرة الثنائية

3-10-3 ارتفاع العقدة height of node :

يمكننا تعريف ارتفاع العقدة في الشجرة من خلال عدد الحواف في أطول مسار من العقدة نزولا إلى الورقة leaf.

4-10-3 ارتفاع الشجرة Height of a tree :

هو عبارة عن عدد الحواف في مسار من الجزر نزولا إلى الورق leaf ,
وارتفاع الشجرة ل n عقدة هي :$\Theta(\lg n)$ which is $\lfloor \lg n \rfloor$, هذا يعني
أن n-عنصر في الكومة ارتفاعها هو $\lfloor \lg n \rfloor$

لاظهار ذلك افرض ارتفاع الشجرة h وبناءا على اكبر واقل عدد من العناصر سوف نحصل على أن :

$$2^h \le n \le 2^{h+1}-1$$

حيث أن n هو عدد العناصر في الكومة .

$$2^h \le n \le 2^{h+1}$$

بأخذ اللوغريتم للأساس 2

$$h \leq \lg n \leq h+1$$

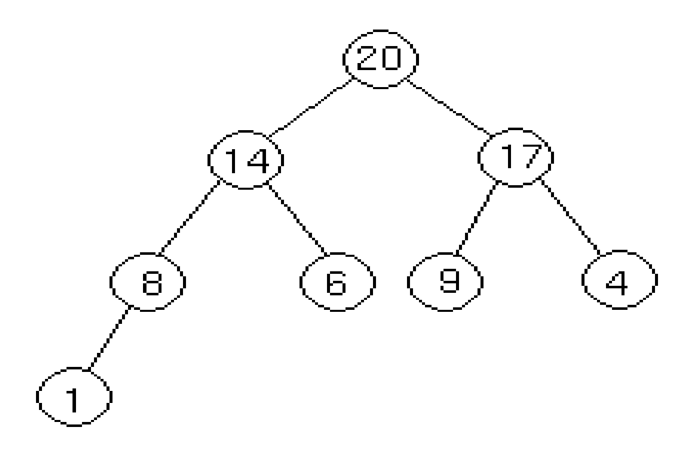
وهذا يقودنا إلى أن $h = \lfloor \lg n \rfloor$

5-10-3 اصغر عنصر في الكومة:

عرفنا من المعطيات أعلاه أن أكبر عنصر في الكومة موجود في الورقة.لجزر , لكن السؤال الذي يطرح نفسه أين يمكن أن نجد العنصر الأصغر في الكومة ؟ بناءا على أي مسار في الشجرة انطلاقا من العنصر الجزر إلى الورقة . ومن خلال خصائص الكومة ,إذا تتبعنا المسار ,فأن قيمة العناصر تكون متناقصة أو ثابتة , فأن الكومة إما إن تكون جميع عناصرها متساوية وبالتالي فأن الشجرة بكاملها هي عبارة عن اصغر أو إن الشجرة يمكن تقسيمها إلى أشجار فرعية والعنصر الأخير هو اصغر عنصر في الشجرة الفرعية. وبصورة عامة العنصر الأصغر في الشجرة هو اصغر عنصر من خلال العنصر الأصغر في جميع الأشجار الفرعية.

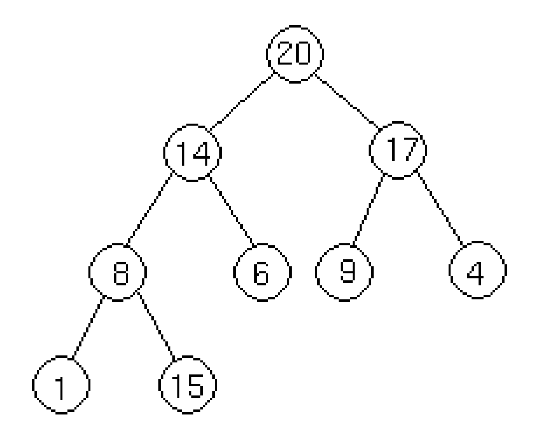
6-10-3 إضافة عنصر في الكومة Inserting Element in the Heap :

بافتراض إن لدينا الكومة التالية:



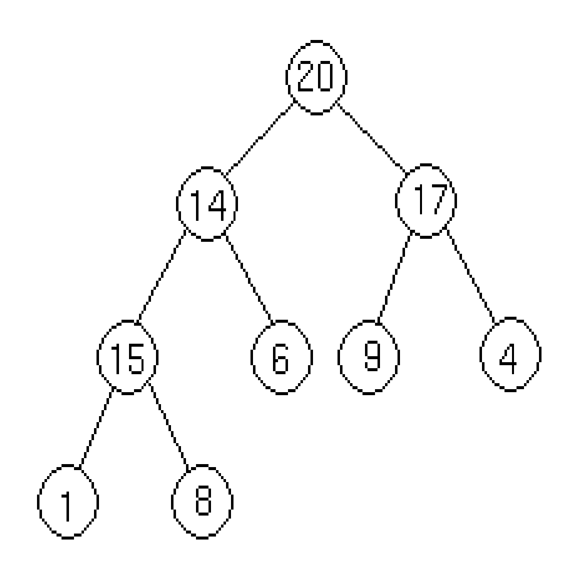
شكل (3-7) إضافة عنصر في الكومة (1)

بافتراض إننا نريد إضافة عقدة بالمفتاح 15 لهذه الكومة, أولا نقوم بإضافة عقدة في الخانة التالية المتاحة في المستوى الأدنى ,مع مراعاة إن تكون شجرة ثنائية تامة.



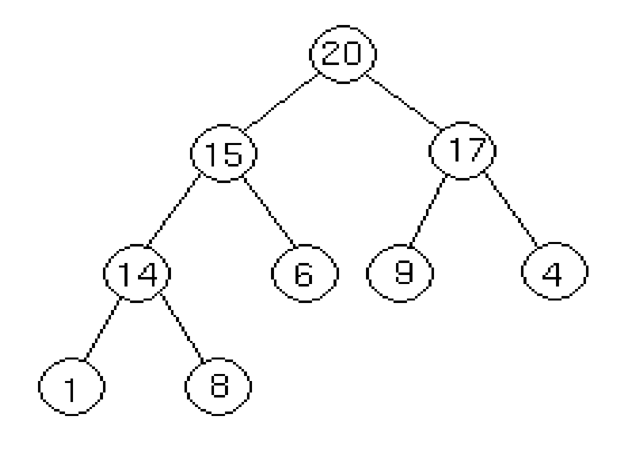
شكل (3-8) إضافة عنصر في الكومة (2)

بعد ذلك نقوم بمقارنة العنصر في العقدة الجديدة المضافة مع العنصر في العقدة الأب لهذه العقدة .فإذا كان العنصر أكبر من أو العنصر في العقدة الأب نقوم بتبديل قيمة العنصر في العقدة الجديدة مع قيمة العنصر في العقد الأب.



شكل (3-9) إضافة عنصر في الكومة(3)

نكرر الخطوة السابقة مع العقدة الأب .



شكل (3-10) إضافة عنصر في الكومة(4)

الآن, تمت إضافة العنصر في مكانه لان العنصر 15<20 وبالتالي لا يوجد تبديل.

3-10-7 إجراءات (دوال) خوارزمية الكومة Heap:

Four basic procedures on heap are:

1. Heapify, which runs in O(lg *n*) time.
2. Build-Heap, which runs in linear time.
3. Heap Sort, which runs in O(*n* lg *n*) time.
4. Extract-Max, which runs in O(lg *n*) time.

3-10-8 وصف الإجراء Heapify :

يختار الإجراء Heapify الابن الأكبر ويتم مقارنته مع الأب, إذا كان الأب اكبر من Heapify يتم الخروج.خلاف ذلك يتم استبدال المفتاح الأب مع الابن الأكبر . ليصبح الأب اكبر من الابن.

ملحوظة مهمة: هذه العملية قد تؤدي إلى إفقاد الكومة خاصية أن الجزر جزر الشجرة الرعية هو اكبر قيمة في الشجرة الفرعية .وفي هذه الحالة يقوم الإجراء Heapify باستدعاء نفسه مرة أخرى مستخدما اكبر قيمة للعقد الأبناء كجزر جديد.

3-10-8-1 خوارزمية Heapify :

Heapify (*A*, *I*)

1. $l \leftarrow$ left [*I*]
2. $r \leftarrow$ right [*I*]
3. if $l \leq$ heap-size [*A*] and $A[l] > A[I]$

4. then largest ← l

5. else largest ← I

6. if $r \leq$ heap-size [A] and $A[I] > A$[largest]

7. then largest ← r

8. if largest $\neq I$

9. then exchange $A[I] \leftrightarrow A$[largest]

10. Heapify (A, largest)

9-10-3 تحليل الخوارزمية Analysis of Heap sort :

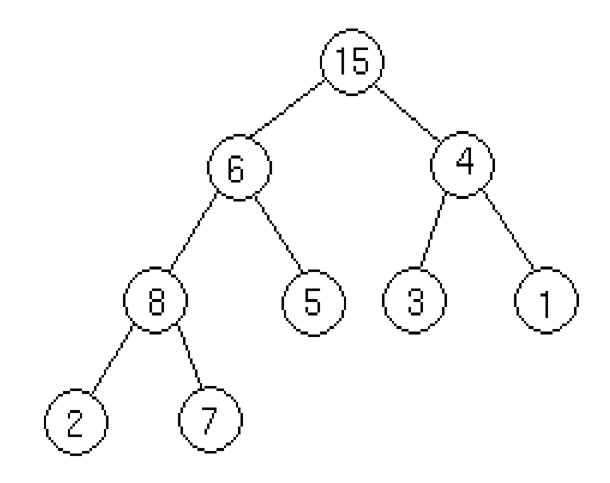
إذا وضعنا قيمة في الجزر اقل من كل القيم في يمين ويسار الشجرة الفرعية ,سوف يتم استدعاء الإجراء 'Heapify' بشكل متكرر (نداءا ذاتيا) حتى الوصول إلى الورق .ولجعل لنداء الذاتي يتتبع المسار الأطول للوصول إلى الورقة, اختر القيمة التي تجعل الإجراء 'Heapify' دائماً يختار الابن الأيسر.وفي هذه الحالة سوف يتتبع الفرع الأيسر إذا كان الابن الأيسر اكبر من أو يساوي الابن الأيمن.مثلا ضع القيمة 0 في الجزر و القيمة 1 في بقية العقد. وفي هذه الحالة لإكمال هذه المهمة سوف يقوم الإجراء 'Heapify' بالاستدعاء h مرة. حيث إن h هو عبارة عن ارتفاع الكومة.

إذن زمن التنفيذ سوف يكون $\theta(h)$.(تقريبا أي استدعاء سوف يتطلب $(\theta(1))$ والذي هو $\Theta(\lg n)$.

وفي هذه الحالة زمن تنفيذ Heapify هي $\Theta(\lg n)$ وهى تمثل الحالة الأسوأ worst-case. إذن زمن التنفيذ هو $\Omega(\lg n)$.

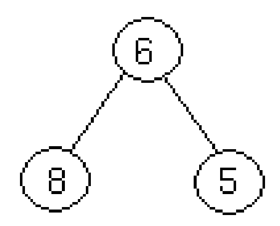
3-10-10 مثال للإجراء Heapify

بافتراض إن لدينا شجرة ثنائية كاملة.وكل شجيرة فرعية هي عبارة عن كومة heap .



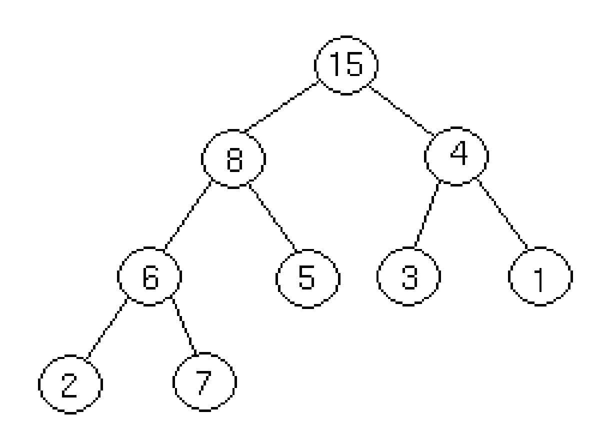
شكل (3-11) مثال للإجراء Heapify (1)

يقوم الإجراء Heapify بتغيير الجزر إلى الموضع 6 . ليصبح جزرا للشجرة وبها طفلان:



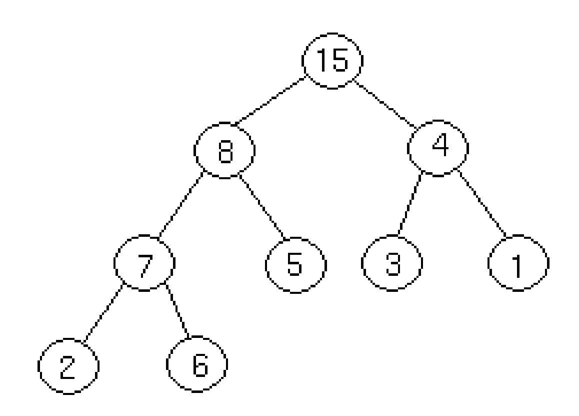
شكل (3-12) مثال للإجراء Heapify (2)

ومن ثم يتم هذه العقد الثلاث هي الأكبر . إذا كانت هي العقدة الجزر معناه تمت لأننا نعامل مع كومة . بخلاف ذلك نقوم باستبدال الابن (الأكبر طبعا) مع العقدة الجزر . ونستمر في تكرار ذلك في نزولا مع الشجرة . وفي المثال السابق نقوم باستبدال 6 مع 8 ونستمر..



شكل (3-13) مثال للإجراء Heapify (3)

حاليا, 7 اكبر من 6 ,نقوم باستبدالهما.



شكل (3-14) مثال للإجراء Heapify (4)

الآن وصلنا إلى أسفل الشجرة ولا نستطيع الاستمرار إذا سوف نتوف هنا.

بناء الكومة Building a Heap :

يمكننا استخدام الإجراء 'Heapify' بصورة معكوسة لتحويل المصفوفة
A[1 . . n] إلى كومة لتصبح جميع العناصر في المصفوفة الفرعية A[$\lfloor n/2 \rfloor$+1 . . n] عبارة عن أوراق leaves . يقوم الإجراء
BUILD_HEAP بالمرور على جميع العقد المتبقية في الشجرة وفي كل
عقدة يستدعي الإجراء 'Heapify' .

هذا الترتيب المعكوس لمعالجة العقد يضمن أن الشجرة الفرعية سوف يكون
لها جزر وأطفال في شكل كومة , قبل استدعاء الإجراء 'Heapify' للأب .

3-10-11 وصف الإجراء BUILD_HEAP :

BUILD_HEAP (A)

1. heap-size (*A*) ← length [*A*]
2. For *I* ← floor(length[*A*]/2 down to 1 do
3. Heapify (*A*, *I*)

يتم بناء الكومة heap لمصفوفة غير برتبة بصور خطية من الزمن
linear time

خوارزمية الترتيب الكومي (المقيد) :Heap Sort Algorithm

تعتبر خوارزمية ترتيب الكومة أفضل من الترتيب بالفرز merge sort
والترتيب بالإدخال insertion sort.

الحالة الأسوأ في خوارزمية الترتيب الكومي هي O(n log n) وتقوم
بالترتيب في الموضع in-place .

تبدأ الخوارزمية باستدعاء الإجراء BUILD-HEAP لبناء الكومة heap في المصفوفة .$A[1 . . n]$. وبالتالي تخزين اكبر عنصر في المصفوفة في الجزر .$A[1]$ويمكن وضعه في مكانه الصحيح عبر استبداله مع العنصر $A[n]$ (آخر عنصر في المصفوفة) .

3-10-12 وصف الإجراء HEAPSORT

HEAPSORT (A)

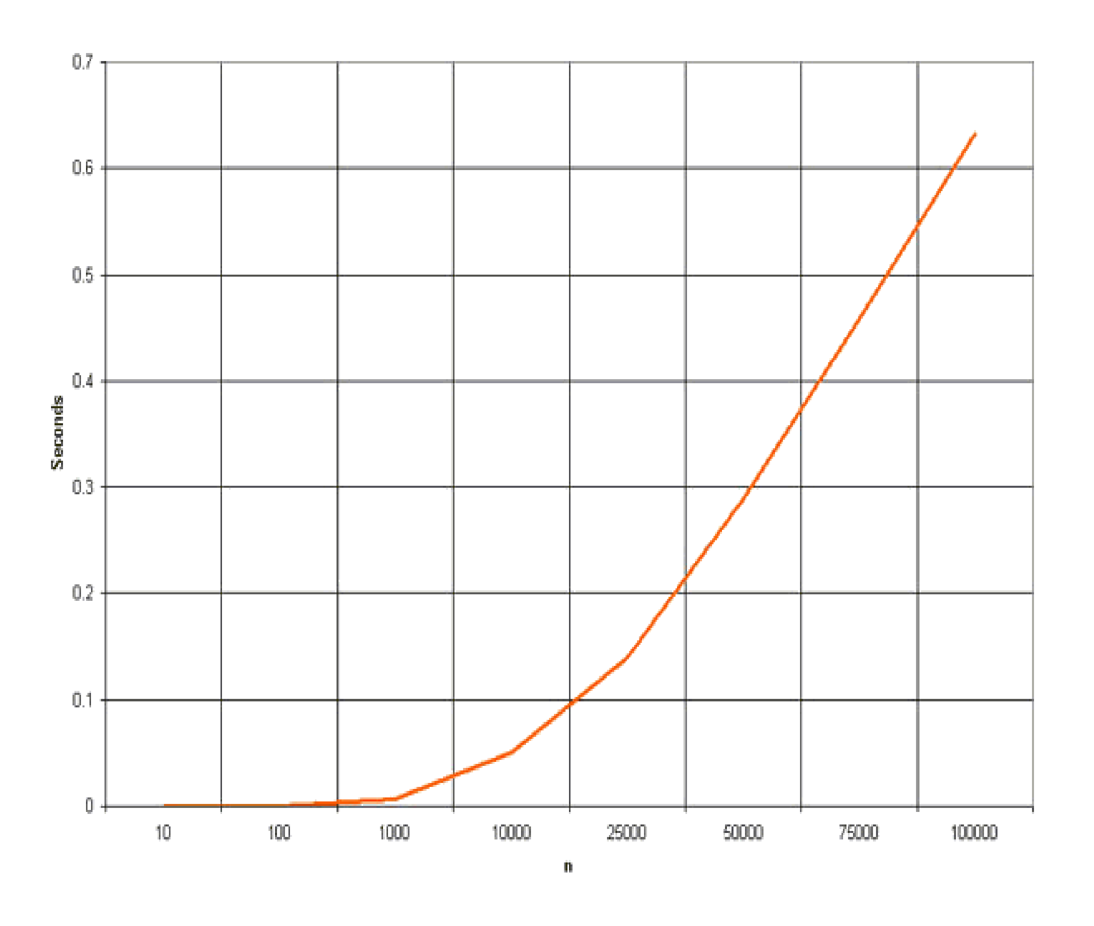
1. BUILD_HEAP (*A*)
2. for *I* ← length (*A*) down to 2 do

   exchange *A*[1] ↔ *A*[*I*]

   heap-size [*A*] ← heap-size [*A*] - 1

   Heapify (*A*, 1)

- الإجراء HEAPSORT يأخذ زمن .$O(n \lg n)$
- في حين إن الإجراء BUILD_HEAP يأخذ زمن O(n)
- وكل n -1 نداء للإجراء to Heapify يأخذ زمن هو O(lg n).



شكل(3-15) مخطط يبين سير تنفيذ خوارزمية الكومة

في الخلاصة, ومن خلال خاصية الشجرة الثنائية فان عدد العقد في أي مستوى هو عبارة عن نصف عدد العقد اعلي هذا المستوى. عدد الورق leaves في الكومة الثنائية يساوي n/2 , حيث إن n هو مجموع العقد في الشجرة إذا كانت عدد العقد زوجي و ⌈ $n/2$ ⌉ إذا كان عدد العناصر فردي . إذا تمت إزالة هذه الأوراق فأن عدد الأوراق الجديدة سوف يصبح$\lg n/2/2$ أو. ⌈ n/4 ⌉

إذا استمرت هذه العملية لعدد h مستوى فأن عدد الأوراق سوف يصبح

⌈ $n/2^{h+1}$ ⌉

13-10-3 إجراءات (دوال) خوارزمية الكومة Heap sort :

```
void heapSort(int numbers[], int array_size)
{
  int I, temp;
  for (I = (array_size / 2)-1; I >= 0; I--)
    siftDown(numbers, I, array_size);
  for (I = array_size-1; I >= 1; I--)
  {
    temp = numbers[0];
    numbers[0] = numbers[I];
    numbers[I] = temp;
    siftDown(numbers, 0, i-1);
  }
}




void siftDown(int numbers[], int root, int bottom)
```

```
{
  int done, maxChild, temp;
  done = 0;
  while ((root*2 <= bottom) && (!done))
  {
    if (root*2 == bottom)
      maxChild = root * 2;
    else if (numbers[root * 2] > numbers[root * 2 + 1])
      maxChild = root * 2;
    else
      maxChild = root * 2 + 1;
    if (numbers[root] < numbers[maxChild])
    {
      temp = numbers[root];
      numbers[root] = numbers[maxChild];
      numbers[maxChild] = temp;
      root = maxChild;
```

```
    }
    else
      done = 1;
  }
}
```

3 -11 Radix sort algorithm :

3-11-1 مفهوم خوارزمية راديكس :

تقوم خوارزمية راديكس radix على مفهوم ترتيب البيانات غير الصحيحة non-integer. حيث تقوم بترتيب البيانات مع مفاتيحها من خلال استخدام الخانات العشرية digits وتستخدم غالبا في ترتيب بيانات غير رقمية (كالأسماء والتواريخ) وأيضاً تستخدم في البيانات التي تحتوي على علامة عشرية floating point .

يعود تاريخ خوارزمية راديس Radix sort algorithm إلى عام 1887 عندما استخدمها Herman Hollerith في آلات الجدولة tabulating machines

معظم الحواسيب الرقمية تمثل البيانات داخليا في صورة رقمية ممثلة في شكل أعداد ثنائية binary numbers لذا معالجة الأعداد الصحيحة في صورة أعداد الثنائية ثنائية يتضمن تصنيفين من الخانات هما :

- least significant digit (LSD)
- most significant digit (MSD)

3-11-2 كفاءة خوارزمية راديكس Radix sort Efficiency

كفاءة خوارزمية راديكس هي: $O(k\cdot n)$ لعدد n مفتاح حيث k هي عدد الخانات العشرية digits .

3-11-3 خطوات الخوارزمية:

Take the least significant digit (or group of bits, both being examples of radices) of each key.

Group the keys based on that digit, but otherwise keep the original order of keys. (This is what makes the LSD radix sort a stable sort).

Repeat the grouping process with each more significant digit.

مثال:

Original, unsorted list:

170, 45, 75, 90, 802, 24, 2, 66

Sorting by least significant digit (1s place) gives:

170, 90, 802, 2, 24, 45, 75, 66

Sorting by next digit (10s place) gives:

802, 2, 24, 45, 66, 170, 75, 90

Sorting by most significant digit (100s place) gives:

2, 24, 45, 66, 75, 90, 170, 802

مثال آخر:

EXAMPLE1:

Here we can sort binary numbers also. Consider a group of 4 bit binary numbers. The list is given by :

1001, 0010, 1101, 0001, 1110

STEP 1:

1st Arrange the list of numbers according to the least significant bit. The sorted list is given by:

0010, 1110, 1001, 1101, 0001

STEP2:

Then arrange the list of numbers according to the next significant bit. The sorted list is given by:

1001, 1101, 0001, 0010, 1110

STEP3:

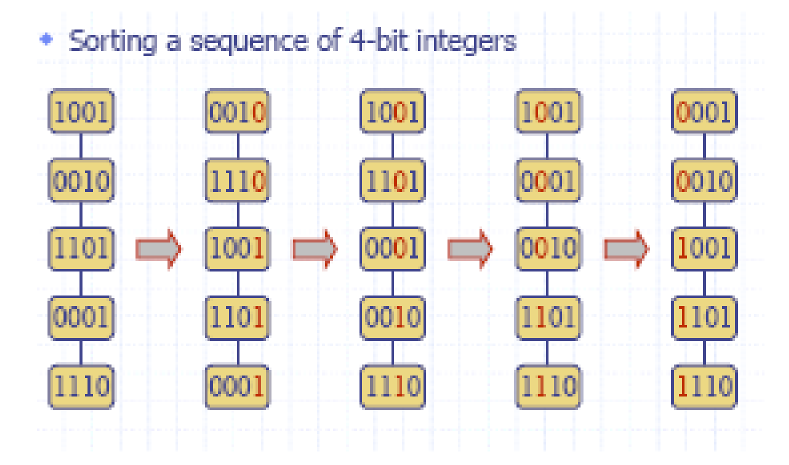
Then arrange the list of numbers according to the 2nd significant bit. The sorted list is given by:

1001, 0001,0010, 1101, 1110

STEP4:

Then arrange the list of numbers according to the most significant bit. The sorted list is given by:

0001, 0010, 1001, 1101, 1110



4-11-3 أجراء خوارزمية راديكس Radix sort procedure

```
void radix(int byte, int size, int *A, int *TEMP) {
  int* COUNT = new (int[256]);
  memset(COUNT, 0, 256 * sizeof(int));
   byte = byte << 3;

  for (int I = 0; I < size; ++I)
   ++COUNT[((A[I]) >> (byte)) & 0xFF];
  for (int I = 1; I < 256; ++I)
   COUNT[I] += COUNT[I - 1];
  for (int I = size - 1; I >= 0; --I) {
   TEMP[COUNT[(A[I] >> (byte)) & 0xFF] - 1] = A[I];
   --COUNT[(A[I] >> (byte)) & 0xFF];
  }
  delete[] COUNT;
}
```

```
void radix_sort(int *A, int size) {
  int* TEMP = new (int[size]);
  for (unsigned int I = 0; I < sizeof(int); I += 2) {
       radix(I, size, A, TEMP);


    radix(I + 1, size, TEMP, A);
  }
  delete[] TEMP;
}
```